Digital Finance – Report

EL ANZI Sofiane - GOV Kévin - LIN Amaury - KUGBLENU Jean Ludovic - MARTINS SARDINHA Alexandre - ZHANG Thierry

Introduction

A decentralized application built on blockchain technology can offer numerous benefits to an insurance car company. DApps can provide transparency and accountability in insurance policies and claims, reducing the risk of disputes and fraud. They are also more secure than traditional software as they are built on a decentralized network of nodes, making it more difficult for a single point of failure or attack. Additionally, DApps have the potential to automate many processes, reducing costs and improving efficiency. By embracing DApps, car insurance companies can improve their operations and better serve their customers.

Car Insurance Company

For this project, we decided to modelized an insurance company by using the blockchain technology.

To explain this project, we will begin by explaining the concept and the utility of the insurance company.

First, a customer submits a claim by filling out an insurance form. There are two kinds of forms:

- A third-party insurance
- An all-risk insurance

When the customer's claim is received by the insurance company, it is first checked by if the claimant has a valid insurance policy. If not, the claimant is informed that the claim is rejected due to an invalid policy.

If the claim is accepted, the claim processing department will register the claim. After the registration, the claim is classified which leads to two outcomes, the third-party or the all-risk.

If the claim is a third-party insurance, the policy is checked, and an assessment is performed.

If the claim is an all-risk insurance, both the policy and the damage are checked independently.

Once the forms are returned, the Claim Handling Department picks up the claim, then an assessment is performed.

For the third party, if the assessment is positive, the Claim Handling Department decides the amount to be paid to the third-party according to the percentage in the expert's report.

For the all-risk, if the assessment is positive, a car dealer is contacted to authorize the repairs and estimate the cost.

When the car dealer finished the reparation, they send a receipt and once the receipt is received, the payment is scheduled.

In any case, whether the outcome is positive or negative, an email is sent to the customer to notify them of the outcome.

Once the claim is processed, the updated claim is registered in the Claim Management System.

Technologies used

For this application, we used Solidity for the backend language which is a high-level programming language used to write smart contracts that run on the Ethereum Virtual Machine (EVM). It is used to write the smart contract that powers the DApp.

We also use Ganache, as a personal blockchain for Ethereum development. It allows developers to create a private blockchain for testing and development purposes. It can be used to create a local blockchain to deploy and test the smart contract.

Moreover, we use MetaMask to interact with Ethereum DApps using the browser. It provides a secure wallet for managing accounts and signing transactions. It can be used to interact with the smart contract using the browser

And for the front-end part, we used Vue JS to modelize the smart contract and their utility through a more understandable application and design.

Functionality

For the smart contracts, we created four smart contracts:

- CarDealer
- Claim_Handling_Dep
- Claim_Processing_Dep
- Customer

We also created tests for the four smart contracts.

The Car Dealer smart contract has the responsibility to:

- Estimate how much the repair is for the client.

The Claim Handling Department smart contract has the responsibility to:

- Set the address of the Car Dealer they want to use for the reparation of their client
- To send the Car Dealer a request for a reparation
- To pay the customer the amount the Car Dealer estimates the reparation costs.

The Claim Processing Department smart contract has the responsibility to:

- Check if the customer is in the database or not
- Check if the insurance of the customer is valid or not
- Check the status of the assessment
- Classify the claim of the customer in All-Risk or Third-Party insurance
- Send a claim to the handling department depending on the category of the risk the customer have

The Customer smart contract has the responsibility to:

- Add a customer
- Update a customer
- Display all the customer

Tests

Each test written on each smart contract has the objective to look at the behavior of the smart contract to see if they are doing the right thing we expected. Each test was successful and correctly behaved according to our expectations.

In the Claim_Handling_Dep_test, we tested two functions "payAmountThirdParty" and "sendClaimDealer".

The first test function sets the expected result to 600 and then calls the "payAmountThirdParty" function of the "claimHandling" contract with a parameter of 50, which returns an actual result. The "Assert" statement then checks if the actual result matches the expected result and displays an error message if they do not match.

The second test function creates a tuple that captures two variables returned by the function, "repairs" and "receipt". It then calls the "sendClaimDealer" function of the "claimHandling" contract with a parameter of 75. The "Assert" statement checks if the "receipt" variable of the returned tuple matches the expected result of 800 and displays an error message if they do not match.

In the CarDealerTest, we tested the only function CarDealer has: "repairs_request".

The first test case checks for the input value of 50. It captures the two variables returned by the function, "repairs" and "receipt", in a tuple. The "Assert" statements check if the "repairs" variable is true, and the "receipt" variable is 600 and displays an error message if they do not match.

The second test case checks for the input value of 75. The same process as the first test case is repeated, and the "Assert" statements check if the "repairs" variable is true, and the "receipt" variable is 800.

The third test case checks for the input value of 90. Again, the same process is repeated, and the "Assert" statements check if the "repairs" variable is true, and the "receipt" variable is 1000.

The fourth test case checks for the input value of 110. The same process is repeated, and the "Assert" statements check if the "repairs" variable is false, and the "receipt" variable is 0.

In the CustomerTest, we tested three function “addCustomer”, “updateCustomer” and “getCustomer”

The first test function adds a new customer to the contract. The function passes in some test values for a new customer, and then retrieves the same customer's data from the contract using the “getCustomer” function. It checks if the values retrieved from the contract match the expected values.

The second test function updates the data of an existing customer. It passes in the ID of the customer to be updated and some new test values, and then retrieves the same customer's data from the contract using the “getCustomer” function. It checks if the values retrieved from the contract match the expected updated values.

The third test function retrieves the data of an existing customer. It retrieves the data of a customer with ID 0 from the contract and checks if the values retrieved from the contract match the expected values.

Development process

Regarding the development process, we started to establish the smart contract and the tests to see how to work with Solidity, Ganache and MetaMask. Then we worked on the front-end part to set up the UI and to display our work. After doing that, we started to work on how we would link the UI and the back end.

Since the deadline was short, we only linked the UI and the back end and fetch some of the function in our back end like the get part, for example, we can get all the customers and their information.

We think that with a much longer deadline, we could have linked all the functionality of our back end with our front end, also redesign the UI to be much more user friendly.

Conclusion

In conclusion, a decentralized application built on blockchain technology offers numerous benefits to an insurance car company, including transparency, accountability, security, and efficiency. By utilizing DApps, car insurance companies can improve their operations and better serve their customers. The DApp we modeled for an insurance company using blockchain technology involved four smart contracts: CarDealer, Claim_Handling_Dep, Claim_Processing_Dep, and Customer, which were tested to ensure they behaved as expected. Solidity was used as the backend language, Ganache as the personal blockchain for Ethereum development, and MetaMask for interaction with Ethereum DApps using the browser. Implementing a DApp for car insurance companies has great potential to revolutionize the industry and offer customers better services.